# Storage engines in mongodb

ANJU MUNOTH

# Storage Engines

The storage engine is the component of the database that is responsible for managing how data is stored, both in memory and on disk.

MongoDB supports multiple storage engines, as different engines perform better for specific workloads.

Choosing the appropriate storage engine for your use case can significantly impact the performance of your applications.

# Storage Engines

**WiredTiger Storage Engine (Default)** -WiredTiger is the default storage engine starting in MongoDB 3.2.

well-suited for most workloads and is recommended for new deployments.

Provides a document-level concurrency model, checkpointing, and compression, among other features.

**In-Memory Storage Engine** -In-Memory Storage Engine is available in MongoDB Enterprise.

Rather than storing documents on-disk, it retains them in-memory for more predictable data latencies.

Starting in version 4.2, MongoDB removes the deprecated MMAPv1 storage engine

# Document Level Concurrency

WiredTiger uses document-level concurrency control for write operations.

As a result, multiple clients can modify different documents of a collection at the same time.

For most read and write operations, WiredTiger uses optimistic concurrency control.

WiredTiger uses only intent locks at the global, database and collection levels.

When the storage engine detects conflicts between two operations, one will incur a write conflict causing MongoDB to transparently retry that operation.

Some global operations, typically short lived operations involving multiple databases, still require a global “instance-wide” lock.

Some other operations, such as dropping a collection, still require an exclusive database lock.

# Snapshots and Checkpoints

WiredTiger uses MultiVersion Concurrency Control (MVCC).

At the start of an operation, WiredTiger provides a point-in-time snapshot of the data to the operation.

A snapshot presents a consistent view of the in-memory data.

When writing to disk, WiredTiger writes all the data in a snapshot to disk in a consistent way across all data files.

The now-durable data act as a checkpoint in the data files.

The checkpoint ensures that the data files are consistent up to and including the last checkpoint; i.e. checkpoints can act as recovery points.

# Snapshots and Checkpoints

Starting in version 3.6, MongoDB configures WiredTiger to create checkpoints (i.e. write the snapshot data to disk) at intervals of 60 seconds.

In earlier versions, MongoDB sets checkpoints to occur in WiredTiger on user data at an interval of 60 seconds or when 2 GB of journal data has been written, whichever occurs first.

During the write of a new checkpoint, the previous checkpoint is still valid.

As such, even if MongoDB terminates or encounters an error while writing a new checkpoint, upon restart, MongoDB can recover from the last valid checkpoint.

The new checkpoint becomes accessible and permanent when WiredTiger's metadata table is atomically updated to reference the new checkpoint.

Once the new checkpoint is accessible, WiredTiger frees pages from the old checkpoints.

Using WiredTiger, even without journaling, MongoDB can recover from the last checkpoint; however, to recover changes made after the last checkpoint, run with journaling

# Journal

- WiredTiger uses a write-ahead log (i.e. journal) in combination with checkpoints to ensure data durability.
- The WiredTiger journal persists all data modifications between checkpoints.
- If MongoDB exits between checkpoints, it uses the journal to replay all data modified since the last checkpoint.
- WiredTiger journal is compressed using the snappy compression library.
- To specify a different compression algorithm or no compression, use the storage.wiredTiger.engineConfig.journalCompressor setting.
- If a log record less than or equal to 128 bytes (the mininum log record size for WiredTiger), WiredTiger does not compress that record.

# Memory Use

- With WiredTiger, MongoDB utilizes both the WiredTiger internal cache and the filesystem cache.
- Starting in MongoDB 3.4, the default WiredTiger internal cache size is the larger of either:

  **50% of (RAM - 1 GB), or 256 MB.**

- For example, on a system with a total of 4GB of RAM the WiredTiger cache will use 1.5GB of RAM (0.5 * (4 GB - 1 GB) = 1.5 GB). Conversely, a system with a total of 1.25 GB of RAM will allocate 256 MB to the WiredTiger cache because that is more than half of the total RAM minus one gigabyte (0.5 * (1.25 GB - 1 GB) = 128 MB < 256 MB).

# Compression

- By default, WiredTiger uses Snappy block compression for all collections and prefix compression for all indexes.
- Compression defaults are configurable at a global level and can also be set on a per-collection and per-index basis during collection and index creation.

# Data in the WiredTiger internal cache versus the on-disk format:

- Data in the filesystem cache is the same as the on-disk format, including benefits of any compression for data files.
- The filesystem cache is used by the operating system to reduce disk I/O.
- Indexes loaded in the WiredTiger internal cache have a different data representation to the on-disk format, but can still take advantage of index prefix compression to reduce RAM usage.
- Index prefix compression deduplicates common prefixes from indexed fields.
- Collection data in the WiredTiger internal cache is uncompressed and uses a different representation from the on-disk format.
- Block compression can provide significant on-disk storage savings, but data must be uncompressed to be manipulated by the server.

# Change Standalone to WiredTiger

- Procedure

1. Start the mongod you wish to change to WiredTiger.If mongod is already running, you can skip this step.
2. Export data using mongodump.

- **mongodump --out <exportDataDestination>**
- Specify additional options as appropriate, such as username and password if running with authorization enabled.

3. Create a data directory for the new mongod running with WiredTiger.

- mongod must have read and write permissions for this directory.
- mongod with WiredTiger will not start with data files created with a different storage engine.

# Change Standalone to WiredTiger

4. Start mongod with WiredTiger.

- Start mongod, specifying wiredTiger as the --storageEngine and the newly created data directory for WiredTiger as the --dbpath.
- Specify additional options as appropriate, such as --bind_ip, and remove any MMAPv1 Specific Configuration Options.

**mongod --storageEngine wiredTiger --dbpath <newWiredTigerDBPath> --bind_ip localhost,<hostname(s)|ip address(es)>**

5. Upload the exported data using mongorestore.

**mongorestore <exportDataDestination>**

Specify additional options as appropriate

How frequently does WiredTiger write to disk?

## Checkpoints

- Starting in version 3.6, MongoDB configures WiredTiger to create checkpoints (i.e. write the snapshot data to disk) at intervals of 60 seconds. In earlier versions, MongoDB sets checkpoints to occur in WiredTiger on user data at an interval of 60 seconds or when 2 GB of journal data has been written, whichever occurs first.

## Journal Data

- MongoDB writes to disk according to the following intervals or condition:
- MongoDB syncs the buffered journal data to disk every 50 milliseconds (Starting in MongoDB 3.2)
- If the write operation includes a write concern of j: true, WiredTiger forces a sync of the WiredTiger journal files.
- Because MongoDB uses a journal file size limit of 100 MB, WiredTiger creates a new journal file approximately every 100 MB of data. When WiredTiger creates a new journal file, WiredTiger syncs the previous journal file.

# How to reclaim disk space in WiredTiger?

- The WiredTiger storage engine maintains lists of empty records in data files as it deletes documents.
- This space can be reused by WiredTiger, but will not be returned to the operating system unless under very specific circumstances.
- The amount of empty space available for reuse by WiredTiger is reflected in the output of db.collection.stats() under the heading wiredTiger.block-manager.file bytes available for reuse.

- To allow the WiredTiger storage engine to release this empty space to the operating system, you can de-fragment your data file. This can be achieved using the compact command

# How to check the size of a collection

- To view the statistics for a collection, including the data size, use the db.collection.stats() method from the mongo shell.
- The following example issues db.collection.stats() for the orders collection:

```
db.orders.stats();
```

MongoDB also provides the following methods to return specific sizes for the collection:

- db.collection.dataSize() to return the uncompressed data size in bytes for the collection.
- db.collection.storageSize() to return the size in bytes of the collection on disk storage. If collection data is compressed (which is the default for WiredTiger), the storage size reflects the compressed size and may be smaller than the value returned by db.collection.dataSize().
- db.collection.totalIndexSize() to return the index sizes in bytes for the collection. If an index uses prefix compression (which is the default for WiredTiger), the returned size reflects the compressed size.

The following script prints the statistics for each database:

```
db.adminCommand("listDatabases").databases
.forEach(function (d) {
   mdb = db.getSiblingDB(d.name);
   printjson(mdb.stats());
})
```

The following script prints the statistics for each collection in each database:

```
db.adminCommand("listDatabases").databases.forEach(function
(d) {
   mdb = db.getSiblingDB(d.name);
   mdb.getCollectionNames().forEach(function(c) {
      s = mdb[c].stats();
      printjson(s);
   })
})
```

# In-Memory Storage Engine

- Starting in MongoDB Enterprise version 3.2.6, the in-memory storage engine is part of general availability (GA) in the 64-bit builds.
- Other than some metadata and diagnostic data, the in-memory storage engine does not maintain any on-disk data, including configuration data, indexes, user credentials, etc.

- By avoiding disk I/O, the in-memory storage engine allows for more predictable latency of database operations.

# Specify In-Memory Storage Engine

- To select the in-memory storage engine, specify:
- inMemory for the --storageEngine option, or the storage.engine setting if using a configuration file.
- --dbpath, or storage.dbPath if using a configuration file.
- Although the in-memory storage engine does not write data to the filesystem, it maintains in the --dbpath small metadata files and diagnostic data as well temporary files for building large indexes.
- For example, from the command line:
- **mongod --storageEngine inMemory --dbpath <path>**

# Concurrency

- The in-memory storage engine uses document-level concurrency control for write operations.
- As a result, multiple clients can modify different documents of a collection at the same time.

# Memory Use

- In-memory storage engine requires that all its data (including indexes, oplog if mongod instance is part of a replica set, etc.) must fit into the specified --inMemorySizeGB command-line option
- By default, the in-memory storage engine uses 50% of physical RAM minus 1 GB.
- If a write operation would cause the data to exceed the specified memory size, MongoDB returns with the error:
  - "WT_CACHE_FULL: operation would overflow cache“
- To specify a new size,

  **mongod --storageEngine inMemory --dbpath <path> --inMemorySizeGB <newSize>**

# Durability

- The in-memory storage engine is non-persistent and does not write data to a persistent storage.
- Non-persisted data includes application data and system data, such as users, permissions, indexes, replica set configuration, sharded cluster configuration, etc.
- As such, the concept of journal or waiting for data to become durable does not apply to the in-memory storage engine.

# Deployment Architectures

- In addition to running as standalones, mongod instances that use in-memory storage engine can run as part of a replica set or part of a sharded cluster.
- Replica Set
- Can deploy mongod instances that use in-memory storage engine as part of a replica set. For example, as part of a three-member replica set, you could have:

**two mongod instances run with in-memory storage engine.**

**one mongod instance run with WiredTiger storage engine. Configure the WiredTiger member as a hidden member (i.e. hidden: true and priority: 0).**

# Deployment Architectures

- With this deployment model, only the mongod instances running with the in-memory storage engine can become the primary.
- Clients connect only to the in-memory storage engine mongod instances.
- Even if both mongod instances running in-memory storage engine crash and restart, they can sync from the member running WiredTiger.
- The hidden mongod instance running with WiredTiger persists the data to disk, including the user data, indexes, and replication configuration information.
- NOTE - In-memory storage engine requires that all its data (including oplog if mongod is part of replica set, etc.) fit into the specified --inMemorySizeGB command-line option